Timer A Module (II)

ReadMeFist

Lab Documents

1) ReadMeFirst
2) Timer Module
3) PinOutSummary
4) InterruptsVectorTable

Overview

This lab consists of one section. We are going to study the capture mode of the timer and use it for frequency measurements.

Background

In this lab we will experiment with the capture mode of Timer A. In this mode we connect an input signal to a Capture Compare Input (CCI) pins (CCIxA, CCIxB etc. in the figure below). When a rising or falling edge (or both) event occurs at a Capture Compare Input, the current value in TAR (Timer A Register) gets recorded into the Capture Compare Register and an interrupt flag CCIFG is set. In other words this mode records the “time” (value in the TAR) in the CCR when an event (rising edge, falling edge etc.) happens at a Capture Compare Input. Thus this mode is often used to measure the width of pulses, time periods of signals (i.e., frequency) etc.

In this experiment we are going to use the capture mode to measure the time period (therefore frequency) of an external signal originating from a signal generator. We can record the “time” (TAR value) of consecutive rising edges of an incoming square wave signal to determine the time period of the square wave. This method can be used, for example, to measure the rotational speed of a motor shaft by measuring the time difference between pulses if the shaft is fitted with sensors that produce pulses.

In this lab we will pay particular attention to the accuracy of the measured time period (or frequency). Different methods will be adopted to deal with high and low frequency measurements respectively.
We are going to start with a summary of the capture mode and then proceed on to the experiments.

**Capture Mode**

On the partial block diagram of Timer A module above, you can see that the capture mode has four different inputs (marked out in the blue square) called CCIXA (Capture Compare Input x A), CCIXB (Capture Compare Input x B), GND and VCC (we will not deal with GND and VCC in this lab). Recall that for each Timer A we have three Capture Compare Registers (CCRs), therefore the value of x is 0, 1 or 2. GND and VCC, as their name imply, are the internal ground the supply voltages of the MCU. For most of the CCR modules CCIXA and CCIXB are connected to pins of the MCU so that you can connect an external signal to those pins. However CC10B and CC11B for Timer0 are not connected to external pins of the MCU, as you can see from the table gotten from the device specific sheet of your MCU (table below). The table lists all the available options. Notice (parts highlighted in red) that CC10B of Timer0 is hardwired internally to the ACLK and CC11B of Timer0 is hardwired internally to the output of the Comparator (Comparator is a peripheral on your MCU, you will get to use it in a later lab). This means that CC10B and CC11B for Timer0 are not accessible to external signals. Also note that for Timer1 all the CCIXA and CCIXB inputs are connected to external pins of the MCU and none of the inputs are hardwired to an internal peripheral.
<table>
<thead>
<tr>
<th>Timer</th>
<th>CC Input Name</th>
<th>Description</th>
<th>Port and Pin Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer0</td>
<td>CCI0A</td>
<td>TA0.0</td>
<td>P1.1</td>
</tr>
<tr>
<td></td>
<td>CCI0B</td>
<td>ACLK</td>
<td>/</td>
</tr>
<tr>
<td></td>
<td>CCI1A</td>
<td>TA0.1</td>
<td>P1.2</td>
</tr>
<tr>
<td></td>
<td>CCI1B</td>
<td>CAOUT</td>
<td>/</td>
</tr>
<tr>
<td></td>
<td>CCI2A</td>
<td>TA0.2</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>CCI2B</td>
<td>TA0.2</td>
<td>PinOsc</td>
</tr>
<tr>
<td>Timer1</td>
<td>CCI0A</td>
<td>TA1.0</td>
<td>P2.0</td>
</tr>
<tr>
<td></td>
<td>CCI0B</td>
<td>TA1.0</td>
<td>P2.3</td>
</tr>
<tr>
<td></td>
<td>CCI1A</td>
<td>TA1.1</td>
<td>P1.1</td>
</tr>
<tr>
<td></td>
<td>CCI1B</td>
<td>TA1.1</td>
<td>P2.2</td>
</tr>
<tr>
<td></td>
<td>CCI2A</td>
<td>TA1.2</td>
<td>P2.4</td>
</tr>
<tr>
<td></td>
<td>CCI2B</td>
<td>TA1.2</td>
<td>P2.5</td>
</tr>
</tbody>
</table>

In order to use the capture mode, you need to

1) Setup Timer A Clock
2) Select capture input source
3) Enable capture mode
4) Select capture logic (rising edge/falling edge or both)
5) Chose synchronous/asynchronous (choose synchronous in this lab)
6) Setup interrupts

**Experiment – Low frequency measurements**

The high/low frequency measurements we are referring here are compared with the Timer A Clock frequency. In this experiment we are going to measure frequencies that are lower than $1/T_{ACLK}$.

Idea of time period measurement is to capture consecutive rising edges/falling edge of the signal connected to CCIxA in this experiment as shown in the figure below.
Recall that in capture mode you are able to record the TAR value at the rising/falling edges. Assume that CCR reads N1 at the first rising edge, and reads N2 at the next rising edge (i.e. after a period), then the period $T_{signal}$ is:

$$T_{signal} = (N2 - N1) \times T_{TACLK}$$

Determine the values of N2-N1 in your ISR (Interrupt Service Routine). Use your own logic in the ISR to make sure that you measure exactly one period. Here is one way you can implement this logic:

Use a register or variable as a flag. Set the value of this flag to 0 at the start of the program. In your ISR, somewhere in the end, toggle this flag. Before the toggling code construct an if statement based on the value of the flag as shown in the pseudo code below:

```pseudo
if(flag == 0)
{
    Measure CCR. This is N1. Store in a register or variable
}
else // flag == 1
{
    Measure CCR. This is N2.
    Calculate N2 – N1 and store this value in a register or variable.

    **Clear CCIFG flag in TA0CCTL0 (explained later in this document)**
}
```

Toggle flag

You can use, for example, BIT0 of R14 register to be the flag.

As stated before, for this method to be accurate we need $f_{TACLK}$ to be much larger than $f_{signal}$.

If we want to be able to measure a frequency as low as 0.1Hz, what clock configuration should you use? Note that we want to calculate the signal period by using
subtraction N2 - N1; for the subtraction to work properly in two’s compliment, N2-N1 cannot be larger than $2^{16} - 1$. Therefore the maximum $T_{\text{TACLK}}$ we can use is limited by the lowest frequency we need to measure (0.1Hz = 10sec). We have the following constrains:

$$(2^{16}-1)T_{\text{TACLK}} > T_{\text{signal}}$$

$$f_{\text{TACLK}} < 6553Hz$$

In this lab, we are going to choose $f_{\text{TACLK}}$ of about 5000Hz.

Use the following settings for setting your Timer A clock:

```plaintext
bic.b #RSEL0+RSEL1+RSEL2+RSEL3, &BCSCTL1
bic.b #DCO2+DCO1+DCO0, &DCOCTL
bic.b #DIVS_2, &BCSCTL2
bic.w #ID_3, &TACTL
```

- Measure and record your SMCLK clock frequency. What is the Timer A clock frequency?
- Use your measured clock frequency to evaluate your result in the report
- Include a screenshot of SMCLK frequency

Use Timer A0 for this experiment and set it to be sourced by SMCLK and run in continuous mode.

Configure Pin1.1 to be used as CCI0A pin

Connect input signal to Pin 1.1.

Configure corresponding CCR to be used in capture mode. Select capture source to CCI0A by choosing correct CCIS bits in TA0CCTL0 register. Configure capture on rising edge.

Enable interrupts so that an interrupt is thrown whenever there is a rising edge event at CCI0A.

Use a signal generator to send in a square wave signal with a **minimum of 0 volts and a maximum of no more than 3.5 volts**. Measure the frequency of this signal using the method and accuracy outlined above. Repeat the experiment for 1Hz, 100Hz, 1kHz, 5kHz. Compare your experimental values with the actual readings on the signal generator. Also calculate the percentage error in your measurement.
How to inspect your results: (Very Important)

Since our microcontroller does not support run time debugging (the CCS debugger cannot update values while the microcontroller is running), we need to set a break point to check our results.

The break point should be set after we have performed the subtraction $N_2 - N_1$. However we will need to manually clear the flag CCIFG in the TAxCCTLx register in order for the break point to work properly even if you are using CCR0 (normally we do not need to clear this flag manually). This is because while your program is paused, the next rising edges of the incoming signal are still being captured by hardware (flag will be raised) causing a wrong TAR value to be registered at wrong time.

---

- Include comparison and analysis of microcontroller measurement results in the lab report.

**Discussion Questions**

There is a limit on how high a frequency you can measure using the method given above: $f_{signal} << f_{TACLK}$. To extend this range of measurement we need to switch to higher clock frequencies at appropriate time. Therefore this condition translates to $f_{signal} << f_{TACLK}$. But what if the frequency to measure is much higher than the frequency of the highest clock frequency that the MCU can offer? i.e.,

$f_{signal} >> f_{clock frequency}$.

In this case we need to follow an alternate method for frequency measurement. To illustrate the method we will use a clock with a low frequency, for example the ACLK, although ACLK is not recommended for timing.

In this method we will drive the timer (TAR) with the input signal whose frequency we want to measure, and we will connect ACLK to the Capture Compare Register module. An illustration of this method is shown in the figure below.
So now we are measuring how many cycles of the signal are contained in one period of ACLK. The period of the signal can be found via:

\[ T_{\text{ACLK}} = (N_2 - N_1) \cdot T_{\text{signal}} \]

Answer the following questions:

1) Which timer and which capture compare register can be used for the high frequency measurement described above using the Launchpad? Give your reasoning.

2) How high a frequency can you measure (assuming all internal timing constrains can be satisfied) by using just the ACLK frequency?